ANGES 1.01

Computational reconstruction of ANcestral GEnomes mapS

Version of August 1, 2014

Cedric Chauve[1], Bradley R. Jones[2], Ashok Rajaraman[3], Eric Tannier[4]

[1]Department of Mathematics, Simon Fraser University, Burnaby (BC), Canada, cedric.chauve@sfu.ca
[2]Department of Mathematics, Simon Fraser University, Burnaby (BC), Canada, brj1@sfu.ca
[3]Department of Mathematics, Simon Fraser University, Burnaby (BC), Canada, arajaram@sfu.ca
[4]UMR CNRS 5558 - LBBE, UCB Lyon 1, Villeurbannes, France, eric.tannier@inria.fr

Contents

1 Overview

ANGES is a suite of Python programs aimed at reconstructing ancestral genome architectures. It follows methodological principles that were described in several recent papers [5, 4, 7].

ANGES requires as input two kinds of data: a *species tree*, annotated to indicate an ancestral species (the *ancestor*[1]), and a set of orthologous *markers* families. The ancestor defines implicitly a partition of the species represented in the species tree into *ingroups* and *outgroups*. The markers represent families of genomic segments that are present in the current extant species and are assumed to have been present, in single-copy, in the ancestor. ANGES infer the organization of these markers in chromosomal segments of the ancestor, i.e. an ancestral genome map, from the comparison of their locations in the current extant species.

A pair of species is said to be *informative* if its evolutionary path contains the ancestor. ANGES proceeds in two main stages.

1. Comparing all informative pairs to detect, for each such pair, genomic segments with a similar organization (in terms of markers) in both compared species (called *Ancestral Contiguous Sets*, or ACS, from now). This is followed by weighting each ACS according to its pattern of occurrences in the considered extant species,
2. Organizing the markers into linear or circular chromosomes by extracting a subset of ACS that satisfies a variant of the Consecutive-Ones Property (*C1P* from now).

We refer to [5, 4, 7] for detailed descriptions of this approach.

This general approach can be implemented in several ways depending of the nature of the markers, of the structure of the expected ancestor or on how a C1P subset of ACS can be found. Additional steps can be also performed that will be described later.

The general principle of our implementation is that every task is implemented into a single Python script, takes for input a set of text files and computes, as output, a set of text files. Hence the scripts that compose ANGES can be organized into a complete ancestral genome inference pipeline (as will be described later) or can be used a stand-alone scripts.

To use the ANGES scripts into a pipeline, ANGES contains a *master script*, that reads the input files (markers and species tree, optional informative pairs and additional ACS) and a *parameters file* that records all user-defined choices that can be made regarding data processing that were described above.

Hence, an *experiment* can be defined by its input files and its parameter file. All files related to an experiment are stored into a unique directory, whose path is indicated in the parameter file. All intermediate computations results are stored into text files within this directory, allowing them to be processed by other programs or scripts.

A simple graphical interface to the master script allows to define a new experiment or to modify some choices of a previous experiment, either to replace it, or to define a new experiment.

In the next sections, we describe in more details the data and method used in ANGES to reconstruct the organization/architecture of an ancestral genome.

2 Input: species, species tree, genomes, ancestor

ANGES requires a fully resolved (i.e. binary) and rooted species tree. This tree is annotated at an internal node to indicate which ancestral species a genome architecture is being reconstructed. This separates the extant species into three well characterized groups: two sets of ingroups and one set of outgroups.

[1] From now, any mention of ancestor or ancestral feature refers to this specific ancestor in the given phylogeny.

Paths to access the markers and species tree file are specified in the parameters file and can be entered through the "I/O" menu of the graphical interface.

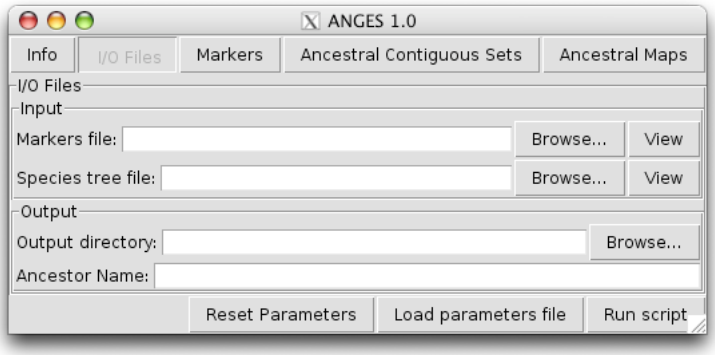

Figure 1: The "I/O Files" window of the graphical interface.

All files computed by ANGES will be placed in a directory indicated in the "Output" window, where the input files and parameters file will also be copied. The structure of this directory is explained later in this manual.

Input genomes can be either multichromosomal with linear chromosomes (for eukaryotic genomes), or circular unichromosomal (chromosomes of prokaryotic genomes). The reconstructed ancestral genome should be of the same kind than the input genomes. Indicating the nature of the ancestral genome is done in the window "Ancestral Maps" of the ANGES interface.

Limitation. Currently, ANGES does not handle circular multichromosomal genomes (such as full prokaryotic genomes with chromosome(s) and plasmid(s)).

3 Input: markers

Markers are families of homologous genomic segments present in ingroup (and possibly outgroup) genomes, that are assumed each to be orthologous to a *unique* (i.e. single-copy) ancestral genomic segment. These markers represent the first level of ancestral reconstruction, and are the basic bricks used by ANGES. Typical families of such markers can be obtained from gene family trees or whole-genome alignment, such as the alignments available on the UCSC Genome Browser and Ensembl websites.

Markers characteristics. As usual with genomic segments, markers are characterized by the physical location(s) (chromosome, genomic coordinates of start and end, orientation) of their occurrence(s) in extant genomes.

Limitation. Currently, ANGES does not accept overlapping markers. All markers occurrences in a given extant genome need to be pairwise disjoint. A script `MARKERS/MARKERS_check_for_overlap_markers` checks if a markers file contains overlapping markers and produce the list of detected overlaps.

Post-processing markers. Once an initial markers file has been selected (called form now the *initial* markers), it can be modified in two ways that we describe below: filtering to keep only a subset of markers or doubling to integrate the orientation of markers in the seeked ancestral genome map. These choices can be don in the "Markers" window.

Uniqueness and universality of markers. Markers can be of several types, depending on the patterns in which they occur in the extant genomes:

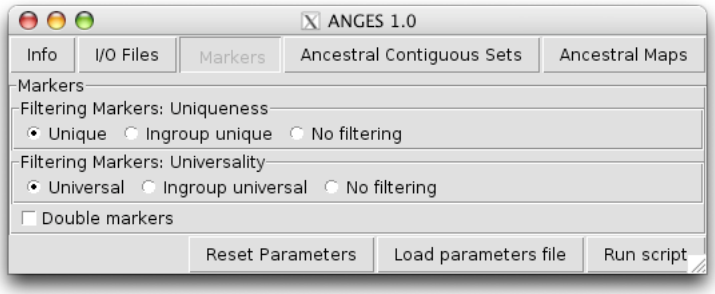

Figure 2: The "Markers" window of the graphical interface.

1. *universal* if each marker occurs at least once in each species,
2. *ingroup-universal* if each marker occurs at least once in each ingroups but might be absent from some outgroup,
3. *unique* if each marker occurs at most once in each species,
4. *ingroup-unique* if each marker occurs at most once in each ingroups but might occur more than once in some outgroup.

Given an initial markers file, ANGES allows to filter it to keep only the markers that satisfies some criteria in terms of uniqueness or universality. The method used in the subsequent steps of ancestral genome map inference can depend on the nature of the selected markers.

By default, unique and universal markers are selected.

Doubling markers. As described in [12], markers can be doubled to be able to infer also their orientation in the ancestor. In such a case, each marker is transformed into two markers: one for each of its two extremities (head and tail in the standard terminology). Hence, in each genome where this marker occurs, this occurrence is replaced by the occurrence of two new markers (each of an arbitrary length of 2bp, as the length of the markers does not impact the subsequent reconstruction steps), called *extremity markers* of an initial marker.

Note that, in the subsequent inference process, it is required that both extremity markers markers of an initial marker are kept consecutive in the ancestral genome map.

4 Ancestral Contiguous Sets

After markers, ACS form the second level of ancestral genomic characters, that represent local ancestral features involving a limited number of markers, that are detected due to their conservation in extant genomes.

Formally, an ACS is a set of markers that are contiguous in at least one informative pair, indicating a potential contiguity in the ancestor. Two kinds of ACS can be computed by ANGES: *adjacencies* (sets of two markers) and *common intervals*.

The choices of combinatorial models of ACS, as well as the way to handle non-unique and non-universal markers are crucial in the process of reconstructing an ancestral genome map, as such a map is computed by "assembling" such local genomic features. The ANGES interface has a window dedicated to select among a wide range of options.

Species pairs. By default, ACS are detected as genomic features that are conserved in at least one informative pair of species. However, an alternative list of species pairs (that do not need to be informative)

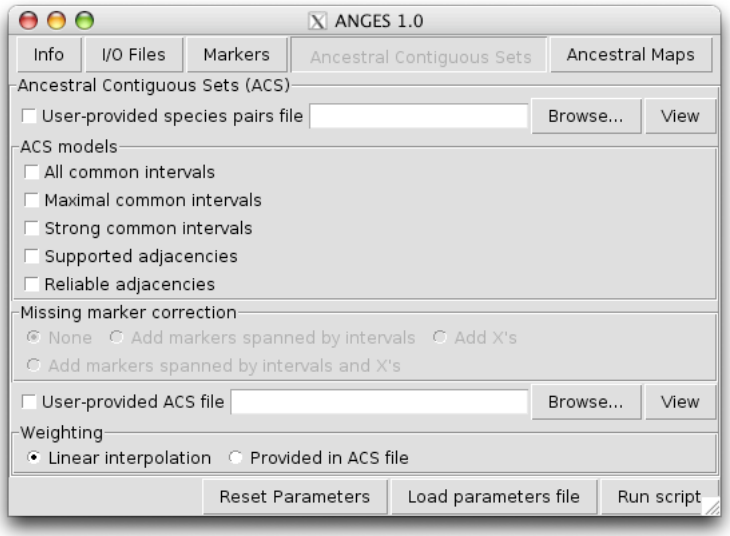

Figure 3: The "Ancestral Contiguous Sets" window of the graphical interface.

to consider can be provided in a separate file (the *pairs* file), in which case only these provided species pairs are considered.

Combinatorial models of ACS: adjacencies. A *supported adjacency* for a given pair of species is a set of two markers that appear consecutively (irrespective of the order) in the genomes of both species.

A *reliable adjacency* is an adjacency that is defined for three input genome such that the ancestor is on the evolutionary path between at least two of them (an *informative triple*). An adjacency is reliable if it belongs to all solutions of the Double-Cut-and-Join median problem for these three genomes (see [4] for a formal definition).

In the case where the initial markers have been doubled, each initial marker is also represented by an adjacency containing its two extremities (i.e. the two new markers defined by its extremities). Such an adjacency is called a *marker adjacency*.

Limitation. Reliable adjacencies can be computed only if the markers are unique and universal, and supported adjacencies have already been computed.

Combinatorial models of ACS: common intervals of linear genomes. A *common interval* for a given pair of species is a set of two or more markers markers that appear contiguously (irrespective of the order) in the genomes of both species. An *occurrence* of a common interval S is a maximal genomic segment whose markers content is the set of markers S. If markers are not unique, a given set S can have several occurrences in a given genome, while it can have only one occurrence if the markers are unique. From now, we define a common interval as a set S of markers[2], that has one or several occurrences in the genomes of a pair of species.

We distinguish three families of common intervals for a given pair of genomes:

1. the set of *all* common intervals;
2. the set of *maximal* common intervals, which are the common intervals having at least one occurrence that is not included into an occurrence of another common interval (used in [5] for example);

[2]Note that, even if markers can have multiple occurrences in some extant genomes (if the markers are non-unique), each marker is assumed to have a unique occurrence in the ancestor, which motivates the representation of ACS as sets of markers and not multisets.

3. the set of *strong* common intervals, which are the common intervals having at least one occurrence that does not overlap an occurrence of another common interval, where two common intervals overlap if their intersection is not empty but none is included (in the classical set-theoretic meaning of inclusion) into the other (see [2] for a formal definition when dealing with unique and universal markers).

The motivation to introduce strong and maximal common intervals is that their number is linear in the size of the considered genomes, while the number of common intervals can be quadratic in the size of the considered genomes. The number of ACS indeed impacts branch-and-bound algorithm in the following stage of assembling ACS into larger chromosomal segments.

Algorithms: linear genomes with unique markers. In the case the markers are not universal, then, prior to computing common intervals, markers that are specific to one of the two genomes of the considered species pair are discarded and the ACS are computed on reduced genomes with equal markers content.

Adjacencies, either supported or reliable, are computed using a trivial linear time algorithm.

If the markers are unique, the model of *common intervals of permutations* is used to compute common intervals. The algorithms used to compute these classes of common intervals are described in [2]. To compute all common intervals, the algorithm we implemented has a time complexity that is linear in the size of the genomes (number of markers) plus the number of common interval, i.e. $O(n + N)$ where n is the number of markers common to both genomes and N the number of common intervals. To obtain only maximal common intervals, then the set of all common intervals is computed then filtered to discard all non-maximal ones. To compute the set of strong common intervals, we follow an optimal algorithm described also in [2], that has a linear $O(n)$ time and space complexity.

Algorithms: linear genomes with non-unique markers. Unlike the case of unique markers, genomes are not reduced to common markers prior to computing ACS.

Supported adjacencies are again computed using a simple linear time algorithm that scans both genomes.

For computing common intervals, we use the framework of *common intervals of sequences*. The algorithms used to compute these classes of common intervals are described in [13] and have a quadratic time and space complexity. All common intervals are computed in time and space $O(n^2)$, where n is the total number of markers in both considered genomes. This set of all common intervals is then filtered to find either all one with at least one maximal occurrence (i.e. an occurrence that is not included into another occurrence of a common interval for the same two genomes) or one non-overlapping occurrence (i.e. an occurrence that is not overlapping another occurrence of a common interval for the same two genomes).

Limitation. The quadratic time and space complexity of the algorithms used to compute common intervals with non-unique markers can result in long computation time, for datasets where genomes are represented with a large number of markers.

ACS for circular genomes. If the considered ancestor is a circular unichromosomal genome, then all extant genomes are assumed to be also circular unichromosomal. The notions of supported adjacency and common intervals have the exact same definitions than in the linear case.

However, the notion of reliable adjacency can not be used as it can be computed only for median genomes that can be multichromosomal. Also, the notions of maximal and strong common intervals do not have, as far as we know, clear definitions in terms of circular genomes, and we rely on ad-hoc definitions for these families of intervals, intended to enforce the important feature of a linear number of maximal or strong common intervals.

The main technical point in computing common intervals and adjacencies of circular genomes is that such genomes are described as linear genomes in most format, with a starting position located, most of the time,

at the origin of replication for bacterial genomes. Hence, the data provided to ANGES do not indicate the circular format (which is indicated by the user in the "Ancestral Maps" window).

To compute supported adjacencies, they are first computed between the linearized genomes, then the only possible missing adjacency, between the first and last markers, is checked, resulting again in a trivial linear time and space algorithm.

If the considered markers are unique, to compute the set of all common intervals, we first compute all (resp. maximal, strong) common intervals between the linearized genomes (reduced to their common marker content as in the linear chromosomes case, with the set of common markers denoted by M), using again the algorithms described in [2], and then, for each such common interval S, we add its complement $M - S$. When computing the set of all common intervals, this process ensures that we obtain all common intervals of the two reduced circular genomes.

If the considered markers are not unique, let G_1 and G_2 be the two considered genomes, linearized. We compute the set of all common intervals between the the genomes $G_1.G_1$ and $G_2.G_2$. i.e. we concatenate a copy of each genome at the end of itself. This ensures that we obtain all common intervals, that are then filtered to discard non-maximal or non-strong ones if required.

Handling non-universal markers. If the considered set of markers is not universal, it can happen that, for a given informative pair, either some markers do not occur in both genomes, or some markers do not occur in one genome. We proceed differently depending if markers are unique or not.

Note: this paragraph was modified on August 1, 2014, to correct a mistake. With unique markers, prior to computing ACS, both genomes are filtered in order to keep only markers that occur in both of them. Given an ACS S computed on these two reduced genomes, missing markers are then handled in three four ways.

0 Missing markers are ignored and the ACS obtained on the two reduced genomes are not changed.

1 If an occurrence of S in a genome defines a genomic segment containing a marker absent from the other genome, this marker is added to the ACS. Hence, if S has several occurrences in the compared genomes, it can be extended into several ACS (at most one per occurrence), depending on the missing markers spanned by its occurrences. This is the option "Add markers spanned by intervals".

2 All markers absent from at least one of the two genomes are added to the ACS as special markers, called X-markers. This is the option "Add X's".

3 Approaches 2 and 3 above are combined: markers missing in both genomes are added as X-markers and markers missing in one genome are possibly added if they are spanned by occurrences of S. This is the option "Add markers spanned by intervals and X's".

With non-unique markers, genomes are not reduced to common markers, and so only approaches 0 and 2 can be applied.

Weighting ACS. To account for the phylogenetic pattern of occurrences of a given ACS, it can be weighted using linear interpolation of a variable along a tree, to produce a weight between 0 and 1, with a greater weight indicating a greater conservation among extant genomes and thus a higher confidence in the ACS as an ancestral character. The algorithm was used in Genomicus [11] and is described in [10].

Formally, for a given ACS, if it is present in a species S, then the leaf corresponding to S in the species tree is assigned a value $w(S) = 1$, and $w(S) = 0$ otherwise. The goal of the weighting is to assign a weight to this ACS at the ancestral node of interest in the species tree. Let N be a node of the (unrooted) species tree; its weight $w(N)$ is 1 or 0 if it is a leaf, and fixed by the occurrence of the considered ACS in N, and, if N is an internal node, $w(N)$ satisfies the equation

$$w(N) = \frac{\sum_{X \text{ neighbor of } N} w(X)\ell_{N,X}}{\sum_{X \text{ neighbor of } N} \ell_{N,X}}$$

where $\ell_{X,N}$ is the length of the branch between N and its neighbor X.

This defines a system of linear equations that can be easily solved using a dynamic programming, that bears similarity with the Fitch-like weighting algorithm that was used in [8, 5].

Telomeric ACS. Here we somewhat abuse of the biological notion of telomeric and we define an ACS as being *telomeric*, for a given species pair, if it occurs in both genomes at the extremity of a chromosome (i.e. there is no marker between the occurrence and the end of the chromosome). Hence the notion of telomeric ACS is purely logical and not related to the genomic and genetic features that define telomeres from a biological point of view.

If telomeric ACS are considered (this is an optional choice that is done in the "Ancestral Maps" window and not in the "ACS" window), then every telomeric ACS is duplicated, with its second copy being augmented by a special marker T indicating the telomeric nature of this ACS.

Limitation. Naturally, telomeric ACS are not relevant when the input genomes are circular. Also, due to algorithmic reasons, ANGES can not handle sets of ACS containing both X-markers and telomeric ACS. Finally, currently, telomeric ACS can be used only with unique markers.

User-provided ACS. The user can provide an additional set of ACS, possibly weighted, encoded into a file provided by the user (format described in Section 4), describing ACS obtained by any other method, that will be appended to the matrix defined by the ACS obtained by comparing informative pair of species. This allows, among others, to re-use a set of ACS whose computation required a long time.

5 Inferring an ancestor from a set of ACS

This last step relies on the Consecutive-Ones Property (C1P) framework, that has been widely used to infer genome physical maps. We review here the principle of this approach and refer to [5, 4, 7] for more details. From a user perspective, there are two main choices to make oin the "Ancestral Maps" window of the ANGES interface: the combinatorial nature of the ancestral genome of interest (circular, linear, linear with telomeres) and the computation method to infer an ancestral genome map from a set of ACS.

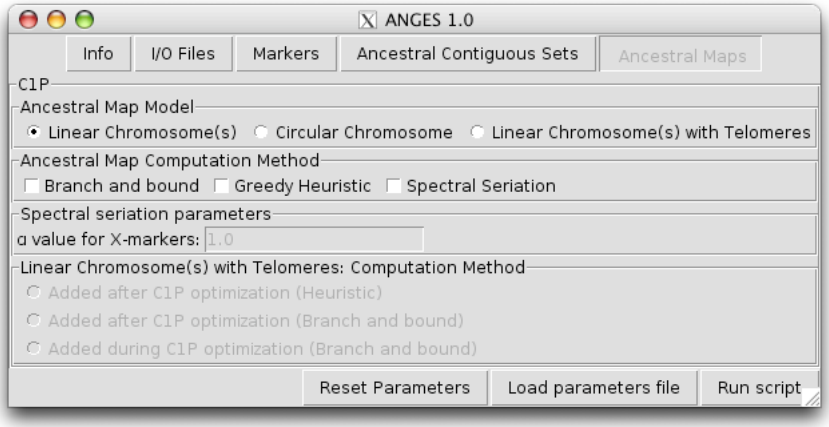

Figure 4: The "Ancestral Maps" window of the graphical interface.

The basic case: binary matrices and linear genomes. A set of ACS, say with no telomeric marker and non X-marker can naturally be seen as a binary matrix: columns of the matrix are markers and each ACS defines a row, with markers present in the ACS defining entries 1 and absent markers entries 0.

If all ACS are true positive (i.e. the set of markers it contains were contiguous in the ancestral genome), then the columns of the matrix can be ordered in such a way that all entries 1 are consecutive on each row: the matrix is C1P. The set of all possible orderings of the markers that satisfy the C1P can be represented by a structure called a *PQ-tree*, that is then a compact way to encode all possible organization of the markers in the ancestral genome. This PQ-tree then defines a set of Contiguous Ancestral Regions (CARs [8]), i.e. putative ancestral chromosomal segments, with possible ambiguities in the marker ordering. More formally, a PQ-tree is a tree with three kinds of nodes: leaves, P-nodes and Q-nodes. The leaves are labeled by the markers in such a way that each marker labels exactly one leaf; P-nodes and Q-nodes are internal nodes, both with a total order on their children. The main property of this tree is that any C1P ordering of the binary matrix can be obtained from the tree by reading, from left to right, its leaf labels after choosing for each node N, independently of the other nodes, (1) an arbitrary order for the children of N if N is a P-node, or (2) to reverse or not the order of the children of N if N is a Q-node (see Fig. 5).

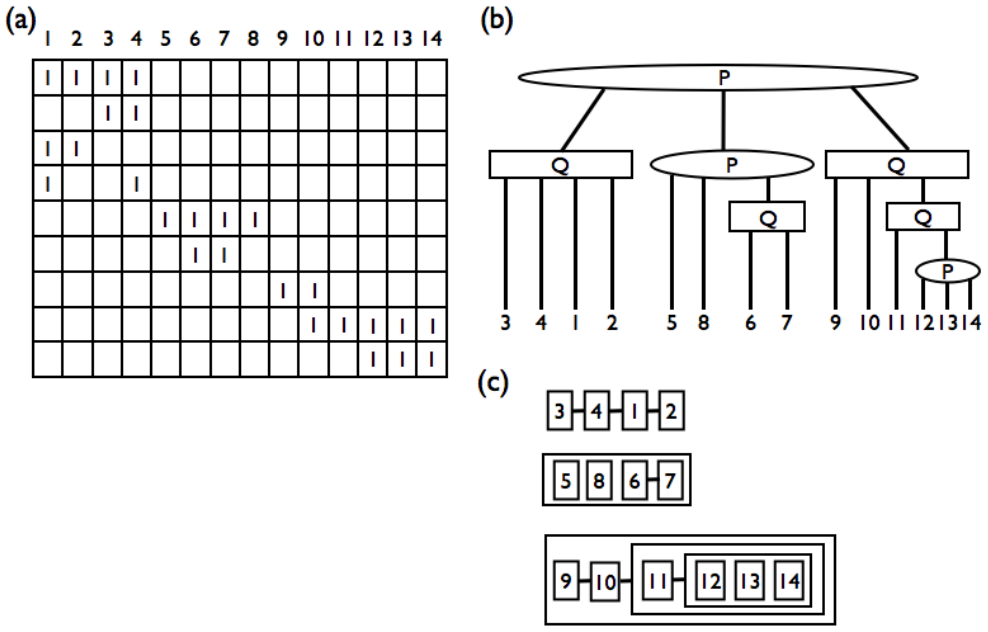

Figure 5: (a) A binary matrix $\mathcal{M}$ with the consecutive ones property, representing a set of 9 ACS on 14 markers. (b) The corresponding PQ-tree $T(\mathcal{M})$, where P-nodes are rounded and Q-nodes are square. 3 4 1 2 5 6 7 8 9 10 11 12 13 14 and 3 4 1 2 9 10 14 12 13 11 6 7 5 8 are two possible C1P orderings for $\mathcal{M}$, among 13824 possible C1P orderings. 3 4 1 2 5 7 6 8 9 10 11 12 13 14 is not a C1P ordering for $\mathcal{M}$: columns 6 and 7 need to be consecutive as they are consecutive children of a same Q-node. (c) An equivalent representation of $T(\mathcal{M})$ which highlights all ancestral genome architectures that correspond to C1P orderings for $\mathcal{M}$: each row corresponds to a chromosomal segment represented by a child of the root, two glued blocks have to be adjacent in any ancestral genome architecture and sets blocks that float in the same box have to be consecutive in any genome architecture but their order is not constrained. Here we see three ancestral chromosomal segments: the first one, which contains markers 1 to 4 is totally ordered; the second one contains markers 5 to 8, with only constraint that markers 6 and 7 are adjacent; the third one contains markers 9 to 14, with 9 and 10 being adjacent, 11 being adjacent to a block that contains 12, 13 and 14 with no order between these three markers. Hence, 9 10 11 12 13 14 is a possible order for this last segment, but not 9 10 12 11 13 14 as 11 is inserted inside the block that contains 12, 13 and 14. All 13824 possible C1P orderings (possible ancestral orderings) are visible on this representation.

In most cases however, the matrix is not C1P, due to, for example, convergent evolution leading to false positive ACS. In this case, we first represent the set of all ACS by a less informative tree, the PQR-tree, where R-nodes are defined by sets of ACS that contain obstruction to the C1P, while the R- and Q-nodes of the PQ-tree indicate subsets of ACS and markers that contain an unambiguous contiguity signal. Formally, such nodes are defined as connected components of the overlap graph of the binary matrix, and we refer to [9, 5] for a precise discussion on PQR-trees (see also Fig. 6).

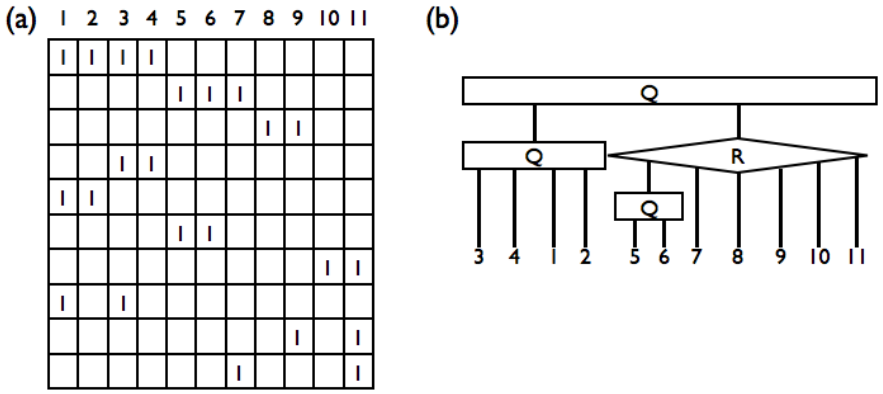

Figure 6: (a) A matrix without the consecutive ones property. (b) The corresponding generalized PQ-tree, where there is a single R-node represented by a diamond shape labeled R. The only R-node is due to the rows 1, 2, 6, 7 and 9 of the matrix that define a sub-matrix that is not C1P, while the submatrix defined by the remaining rows is C1P.

In the case where the set of ACS is not C1P, we aim at selecting a subset of ACS that defines a C1P matrix. We use two methods for this:

1. A greedy heuristic orders the rows of the matrix by decreasing weight and adds them to the subset of ACS (empty at first) if this addition does not violate the C1P (this method was used in [8] for example).
2. A branch-and-bound algorithm computes a subset of ACS that satisfies the C1P variant and whose summed rows weights is maximum. Although this problem is NP-hard, if the seeked subset of ACS is close to the initial one (i.e. few ACS need to be discarded to satisfy the C1P), this approach can find an optimal solution relatively quickly.

In both cases, we obtain two subsets of ACS: the C1P subset (the ones that satisfies the C1P) and the discarded ones (the ones that needed to be discarded in order to satisfy the C1P).

Both approaches are complementary. The greedy heuristic provides an initial bound on the amount of ACS to discard to obtain a C1P matrix. In the case the data encoded in the ACS is close to being C1P, this bound is generally accurate and ensures that the branch-and-bound terminates with an optimal result in a reasonable time. If the greedy heuristic results in a large set of discarded ACS, it is likely that the branch-an-bound will require a long time to terminate, and more importantly, that the signal encoded in the ACS contains a significant amount of false positive, and it makes sense to investigate these ACS to understand the cause of such conflict before using them to infer an ancestral genome map.

The result of this phase is then a PQ-tree representing the inferred ancestral organization of markers into CARs.

Doubled markers. When markers are doubled, the marker adjacencies are weighted in such a way that they are conserved in any subset of ACS computed either through the heuristic or the branch-and-bound. In PQ-trees with such doubled markers, pairs of markers originating from an initial marker are thus consecutive,

indicating an orientation of this initial marker. Such a PQ-tree is then "halved" to be described in terms of the initial markers, ensuring the doubling of markers is transparent.

Variant: circular ancestral genome. If the seeked ancestral genome is circular, then the way to proceed is highly similar. However, we do not aim at selecting a set of ACS that satisfy the C1P, but the circular C1P (*cir-C1P*): we seek an order of the columns of the matrix (markers) such that, for each row, either all entries 1 are consecutive, or all entries 0 are consecutive. The resulting combinatorial structure describing CARs is not a PQ-tree (resp. PQR-tree), but a PQC-tree (resp PQCR-tree): the root node is a C-node indicating a circular ordering of the CARs.

Variant: linear ancestral genome with telomeric markers. If the set of ACS contains telomeric ACS, then the PQ-tree and PQR-tree are augmented by leaves labeled T (the telomeric marker), that need to be located at the extremities of CARs. CARs that contain a telomeric marker at both extremities can then be considered as full ancestral chromosomes. To select a subset of ACS that satisfy the C1P we can use the greedy heuristic or branch-and-bound described above, but ANGES offers an alternative approach, that consists in (1) computing a PQ-tree with no telomeric markers using the branch-and-bound approach described above, followed by (2) computing, again using a branch-and-bound approach, a maximum (in terms of weight again) subset of telomeric ACS that can be integrated while respecting the structure of the previously computed PQ-tree.

Handling ACS with X-markers: ternary matrices and the X-C1P. ACS with X-markers can be encoded into ternary matrices with entries 0 and 1 (as above) and X for the X-markers. The C1P framework can then be extended to see if there is a way to assign to each X entry a value 0 or 1 in such a way that the resulting binary matrix is C1P: this variant foe the C1P is the *sandwich C1P*, called here *X-C1P*. Unlike other variants of the C1P described above, deciding if a ternary matrix is X-C1P is NP-hard, and the method described above (greedy heuristic and branch-and-bound) can not be applied, as they rely on numerous C1P tests.

Currently, ANGES implements the computation of a correlation matrix, parametrized by a parameter α that can is chosen by the user (default value: 1). More, precisely, the correlation between two markers A and B is defined as the sum, over all ACS, of 1 if both A and B belong to the ACS, 0 if both do not belong to it, α if one belongs to the ACS and the other one is an X-marker, and α^2 if both are X-markers. This definition is a natural generalization of the dot-product that was used in [1]. Once such a correlation matrix is computed, it can be processed using the spectral seriation algorithm described in [1], that computes a PQ-tree from the correlation matrix, that is the same PQ-tree representing all C1P-ordering if the set of ACS is C1P [1] or approximates orderings that minimizes gaps of zeros between 1s [14].

Limitation. The spectral seriation approach applies only to linear chromosomes and can not be used with telomeric ACS. Also, as it is a heuristic that can not enforce the contiguity of selected pairs of columns, it can not be applied with doubled markers and doe not result in a set of discarded ACS.

Algorithms. Efficient algorithms used to test the C1P and to compute PQ-trees are taken from [9]: the set of ACS is seen as a binary matrix, connected components of the overlap graph are computed, that define an inclusion tree that serves as backbone of a PQ-tree, and then each connected component is processed using partition refinement. If the set of ACS is not C1P, then, in an initial stage, a PQR-tree [9, 5] is computed that allows to point at sets of ACS that satisfy the C1P and induce well-defined CARs.

Both the branch-and-bound and greedy heuristic are applied independently one each set of ACS defined by a connected component of the overlap graph, which is key to reasonable computation time using the branch-and-bound approach.

When dealing with circular chromosomes, he algorithmic approach we follow, either for the greedy heuristic or the branch-and-bound, relies on a tight link between the C1P and the circ-C1P (see [6] for example).

More precisely, it relies on the well-know property that a binary matrix is circ-C1P if and only if the matrix obtained by (1) picking an arbitrary column and (2) complementing each row/ACS having an entry 1 in this column (i.e., for each such row, replacing 0s by 1s and conversely) is C1P.

Given a set of (non-telomeric) ACS that is C1P, we can augment it by a subset of telomeric ACS in order to satisfy the C1P with multiplicity (*mult-C1P*, as the telomeric marker T can have multiple occurrence, at most two per CAR). This approach was described in [3], where it was shown that deciding the mult-C1P for telomeric ACS is tractable, and we refer the reader to [3] for more details.

6 File formats

6.1 Species Tree file format.

The species tree needs to be in the NH or NHX format, with branch length, no name neither comment at internal nodes, and with an '@' at the location of the ancestor whose genome map is to be computed.

Example: `((Homo_sapiens:0.01562,Pan_troglodytes:0.01562)@:0.0198250,Pongo_pygmaeus:0.03544);` In the example above, the ancestor is the last common ancestor of species `Homo_sapiens` and `Pan_troglodytes`, with one outgroup, `Pongo_pygmaeus`.

6.2 Markers file format.

A marker is defined by a name and the characteristics of its occurrences in the extant species.

```
>marker_name
species.chromosome:start-end [+/-]
species.chromosome:start-end [+/-]
...

...
```

`marker_name` must be an integer and is used to identify the marker. It can be followed by comments describing the family of markers.

`species` is the species name and should not contain a ".", as it would cause problems in parsing the blocks occurrences. Similarly, "chromosome" should not contain the symbol ".".

Example:

```
>1
Homo_sapiens.1:1246289-1887289 +
Pan_troglodytes.1:1238041-1835114 +
Pongo_pygmaeus.1:228608798-229274150 -
Macaca_mulatta.1:4378954-4994782 +
Mus_musculus.4:154816126-155263705 -
Rattus_norvegicus.5:172262731-172747386 -
Equus_caballus.2:47900409-48325808 -
Canis_familiaris.5:59485848-59925615 +
Bos_taurus.16:48071865-48486015 -
Taeniopygia_guttata.21:3815047-4354881 -
Monodelphis_domestica.2:103163810-104032099 +
Gallus_gallus.21:1863772-2350501 -

>2
```

```
Homo_sapiens.1:2330854-2704271 +
Pan_troglodytes.1:2293398-2879717 +
Pongo_pygmaeus.1:227893864-228155669 -
Macaca_mulatta.1:5457661-6023660 +
Mus_musculus.4:154232070-154461601 -
Rattus_norvegicus.5:171660536-171889041 -
Equus_caballus.2:47229693-47495866 -
Canis_familiaris.5:60277278-60482542 +
Bos_taurus.16:47528778-47725916 -
Taeniopygia_guttata.21:3243323-3442778 -
Monodelphis_domestica.4:376497489-377215660 +
Gallus_gallus.21:1342549-1538614 -
```

6.3 Species pairs file format.

The species pairs file format is simple: each line of the file contains a the names of two species defining such a pair.

Example:

```
Homo_sapiens Pan_troglodytes
Homo_sapiens Pongo_pygmaeus
```

In the file above, the informative pair `Pan_troglodytes Pongo_pygmaeus` (see example of species tree, Section 6.1) is not considered.

6.4 ACS file format.

ACS are encoded into files following a generic format aimed at recording evolutionary conserved genomic features. Each feature is characterized by a unique `id` (which can be any value (without spaces, '|', ';', ',' or ':'), a `weight` (aimed at weighting it according to its phylogenetic conservation for example), the `list_of_species` that contain this feature, and finally its description (`information`).

For an ACS, the information starts by the set of markers that define it. If an ACS is telomeric, it is indicated by a special marker `T`. If an ACS contains X-markers, they are indicated by an `X` followed by the list of all X-markers. An ACS can bot be both telomeric and contaning X-markers.

Format:

```
id|weight;list_of_species:list of markers in the ACS
...
```

Example 1 (with telomeric markers):

```
1|0.5;Equus_caballus,Gallus_gallus,Homo_sapiens,Macaca_mulatta:1 2 3 4 5 6 7 8 9
2|0.225;Gallus_gallus,Homo_sapiens,Macaca_mulatta:1 2 3 4 5 6 7 8 9 T
3|1.0;Bos_taurus,Gallus_gallus,Taeniopygia_guttata:1 2 3 4 5 6 7 8
4|0.345;Canis_familiaris,Gallus_gallus:1 2 3 4 5 T
5|0.345;Canis_familiaris,Gallus_gallus:1 2 3 4 5
```

Example 2 (with X-markers)

```
1|0.5;Equus_caballus,Gallus_gallus,Homo_sapiens,Macaca_mulatta:1 2 3 4 5 6 7 8 9
2|0.225;Gallus_gallus,Homo_sapiens,Macaca_mulatta:12 13 X 15 17
3|1.0;Bos_taurus,Gallus_gallus,Taeniopygia_guttata:1 2 3 4 5 6 7 8
4|0.345;Canis_familiaris,Gallus_gallus:1 2 3 4 5 X 12 13
```

6.5 CARs file format.

The ancestral genome organization defined by an ACS file is encoded using a combinatorial structire known as PQR-tree, in a format that is close to the standard one used to encode gene order.

An ancestral genome map is encoded as follows:

```
>species_name
#CAR1
markers_organization along CAR1
#CAR2
markers_organization along CAR2
...
```

`species_name` is the name of the ancestral species whose map is described in the file (the node marked byïn the species tree).

`markers_organization` contains the list of all markers of a CAR, possibly signed (to encode orientation) if this option was chosen by the user, together with *markups* encoding the *nested* structure of these markers. This nested structure is intended to represent possible ambiguities in the order of a set of markers computed as being contiguous in the ancestral genome map.

Markups can encode four kinds of combinatorial organization of markers: R-nodes, P-nodes, Q-nodes and C-nodes. This terminology is borrowed from the mathematical theory of PQR-trees, and should not hide the fact that the organization of markers along CARs is linear or circular. Each type of organization uses two markups: an opening markup (`_R` for R-nodes, `_Q` for Q-nodes, `_P` for P-nodes and `_C` for C-nodes) and an associated closing markup (`R_`, `Q_`, `P_`, `C_`) and the markers that are located between these two related markups are exactly the correspondign combinatorial structure.

Markers belonging to an R-node, i.e. located between an opening markup `_R` and the corresponding closing markup `R_` represent a subset of ACS that is non-C1P. Formally, all sets of ACS that contain only markers located between these two markups form a non-C1P matrix (a non circ-C1P matrix if the ancestral genome map is expected to be circular).

Markers belonging to a Q-node, i.e. located between an opening markup `_Q` and the corresponding closing markup `Q_` are totally ordered according to the order given in the file. Below are two examples:

- `_Q 4 -2 1 5 Q_` represents a CAR with 4 markers in the order 4,2,1,5, with marker 2 on the reverse strand.
- To illustrate the nested structure, the CAR `_Q 4 -2 _R 6 7 _Q 10 -13 Q_ 9 R_ 1 5 Q_` encodes a structure where markers 6,7,10,13 have to be between 2,4 on one side and 1,5 on the other, 10,13 are contiguous (with marker 13 reversed), but the ACS containing only markers from the set 6,7,10,13 are not C1P.

A C-node, defined by markups `_C` and `C_` is a Q-node where the total order is circular: the last marker is assumed to be adjacent to the first one. An ancestral genome map with a C-node can have only one CAR as currently ANGES does not handle multichromosomal circular genomes.

Finally, a P-node, defined by markups `_P` and `P_`, encodes a set of markers that are assumed to be contiguous, but whose order is not resolved: all possible orders are compatible with the set of ACS. For example, `_Q 1 2 _P 4 6 _Q 3 7 8 Q_ P_ 5Q_` encodes 12 possible unsigned orders for the markers 1 to 8:

```
1 2 4 6 3 7 8 5,  1 2 4 6 8 7 3 5,  1 2 6 4 3 7 8 5,  1 2 6 4 8 7 3 5,
1 2 4 3 7 8 6 5,  1 2 4 8 7 3 6 5,  1 2 6 3 7 8 4 5,  1 2 6 8 7 3 4 5,
1 2 3 7 8 4 6 5,  1 2 8 7 3 4 6 5,  1 2 3 7 8 6 4 5,  1 2 8 7 3 6 4 5.
```

Example of a PQ-tree:

```
>Ancestor_1
#CAR1
```

```
_Q_P20 1 3 5 7 10 11 12 14 15 16 17 R_ _P8 13 6 P_ Q_
#CAR2
_P 332 333 _P 334 335 336 P_ 337 _Q 338 339 340 Q_ 341 342 343 P_
```

Example of a PQR-tree:

```
>Ancestor_1
#CAR1
_Q_P20 1 3 5 7 10 11 12 14 15 16 17 R_ _P8 13 6 P_ Q_
#CAR2
_P 332 333 _R 334 335 336 337 _Q 338 339 340 Q_ R_ 341 342 343 P_
```

Example of a PQC-tree: a single CAR that is a C-node.

```
>Ancestor_1
#CAR1
_C _Q_R20 1 3 5 16 17 R_ _P8 13 6 P_ Q_  _P 332 333 _P 334 335 336 P_ 337 341 342 343 P_ C_
```

7 The ANGES pipeline

As described above, the inference of ancestral genomes organization can be implemented in several ways depending on the nature of the markers, input genomes, computation choices, ... ANGES offer an integrated way to deal with this multiplicity of options, through

1. either a command-line approach that reads a parameter file indicating user choices for all possible options (nature of markers, ACS computation, C1P computations),
2. or a simple graphical interface, that allows the user to enter values for all choices, which are recorded into a parameters file, and then analyze the data according to these choices.

The graphical interface allows also to read and modify an existing parameters file. ANGES also checks that no incompatible choices have been made; using the graphical interface ensures such choices can not be made.

A set of choices, recorded into a parameter file,together with input files define an *experiment*. Among others, an experiment is associated to a directory `<EXP_DIR>` whose name is specified in the parameters file, and that will contain all the files generated by the related computations and an ancestor name. All files of an experiment are prefixed by the ancestor name, `<ANCESTOR>`, again specified in the parameters file. These files are placed in several directories.

`<EXP_DIR>/INPUT` contains the parameters file (named `<ANCESTOR>_PARAMETERS`), the species tree file `<ANCESTOR>_SPECIES_TREE`, the markers file `<ANCESTOR>_MARKERS`, and the optional user-provided informative pairs file and ACS file.

`<EXP_DIR>/MARKERS` contains the filtered (by uniqueness and universality criterion) markers file and the optional doubled markers file. The initial markers file is named `<ANCESTOR>_MARKERS` and the doubled markers file `<ANCESTOR>_MARKERS_DOUBLED`.

`<EXP_DIR>/ACS` contains the results of all ACS computations. Files encoding ACS of type `<ACS_TYPE>` (that can be `SA` for supported adjacencies, `RA` for reliable adjacencies, `ACI` for all common intervals, `MCI` for maximal common intervals and `SCI` for strong common intervals) obtained by comparing two species `<SPECIES1>` and `<SPECIES2>` are named `<ANCESTOR>_<ACS_TYPE>_<SPECIES1>_<SPECIES2>`. Moreover, a file recording all ACS between the two species is named `<ANCESTOR>_ACS_<SPECIES1>_<SPECIES2>`.

If missing markers are added, additional files are generated, whose name is obtained by suffixing each ACS file name by either `MM1` if approach 1 is followed (adding markers belonging to intervals spanned by an ACS) or `MMX` if X entries are added (approach 2) or `MM1X` if the combined method is applied (approach 3).

If telomeric ACS are added, an additional file is generated, containing all ACS, and named <ANCESTOR>_<ACS_TYPE>_<SPECIES1>_<SPECIES2>_TEL. Finally, a file containing all ACS (including the telomeric ones if relevant) is named <ANCESTOR>_ACS.

<EXP_DIR>/WEIGHT contains the result of weighting the ACS. It contains a single file <ANCESTOR>_WACS.

<EXP_DIR>/C1P contains the results of the C1P computations, i.e. the selection of subsets of ACS, excluding the PQ-trees and PQC-trees representing the ancestral genome maps. Again, files are suffixed in order to indicate what they represent. For a given ACS file and a given method to select a C1P (or variant) subset of ACS, it computes two files: a file of kept ACS and a file of discarded ACS, named respectively <ANCESTOR>_ACS_C1P_<METHOD> and <ANCESTOR>_ACS_DISC_<METHOD> where <METHOD> method is either BAB for the branch-and-bound or HEUR for the greedy heuristic or SERIATION for the spectral seriation approach. If telomeric ACS are considered, <METHOD> is either TEL_BAB1 for the branch-and-bound in two stages, TEL_BAB2 for the global branch-and-bound or TEL_HEUR for the greedy heuristic.

<EXP_DIR>/CARS contains the ancestral genome maps, represented by PQ-trees and variants. PQR-trees and PQCR-trees are in files <ANCESTOR>_PQRTREE and <ANCESTOR>_PQRTREE_DOUBLED (the former expressed in terms of the initial markers, the later in terms of doubled markers). PQ-trees and PQC-trees are in files <ANCESTOR>_PQTREE_<METHOD> and <ANCESTOR>_PQTREE_DOUBLED_<METHOD> where METHOD is as described in the previous paragraph. All files in this directory follow the CARs file format.

Finally, a log file EXP_DIR/ANCESTOR_LOG keeps track of all steps of the computation, when ANGES is run using the graphical interface.

8 Description of Python scripts

We list below the main scripts available in ANGES. We refer to comments in each script for a description of its parameters, input and output formats.

Pipeline scripts. The two scripts allowing to use ANGES as a pipeline for the computation of an ancestral genome map are the following.

MASTER/anges_CAR.py: the command line script.

MASTER/anges_CAR_UI.py: the graphical interface.

Species trees. TREES/TREES_list_species.py: Writes in a file the names of subsets of extant species from a species tree.

TREES/TREES_list_species_pairs.py: Writes in a file all informative pairs from a species tree.

Markers. MARKERS/MARKERS_double.py: Transforms each marker into two markers, one for each extremity.

MARKERS/MARKERS_filter_species.py: Filter a set of markers to discard the markers that do not satisfy some given condition of uniqueness and universality.

MARKERS/MARKERS_check_for_overlaps_markers.py: Checks if a given markers file contains overlapping markers and produces a list of markers involved in overlaps.

Ancestral Contiguous Sets. `ACS/ACS_compute_supported_adjacencies_unique.py`: Computes all supported adjacencies between two species with unique markers.

`ACS/CS_compute_reliable_adjacencies_unique.py`: Computes all reliable adjacencies between two species. It assumes that supported adjacencies between these two species have been computed.

`ACS/ACS_compute_common_intervals_unique.py`: Computes common intervals between two species when markers are unique.

`ACS/ACS_compute_supported_adjacencies_nonunique.py`: Computes all supported adjacencies between two species with unique markers.

`ACS/ACS_compute_common_intervals_nonunique.py`: Computes common intervals between two species when markers are unique.

`ACS/ACS_add_missing_markers.py`: Add into a set of ACS markers that are present in at most one of the species.

`ACS/ACS_join_files.py`: Concatenate into a single file several ACS files.

`ACS/ACS_add_telomeres.py`: Add telomeric ACS.

`WEIGHT/WEIGHT_weight_linear_interpolation.py`: Weight each ACS using linear interpolation of a variable along a tree.

Consecutive-Ones Property. The main scripts to handle the C1P and its variants re the following.

`C1P/C1P_check_C1P.py`: Checks if a set of ACS satisfies the C1P.

`C1P/C1P_check_circC1P.py`: Checks if a set of ACS satisfies the circ-C1P.

`C1P/C1P_check_mC1P.py`: Checks if a set of ACS with telomeric ACS satisfies the mult-C1P.

`C1P/C1P_compute_PQRtree.py`: Computes a PQR-tree from a set of ACS. If the set of ACS satisfies the C1P, then the tree is a PQ-tree.

`C1P/C1P_compute_PQCRtree.py`: Computes a PQCR-tree from a set of ACS. If the set of ACS satisfies the circ-C1P, then the tree is a PQC-tree.

`C1P/C1P_make_C1P_branch_and_bound.py`: Computes a subset of ACS of maximum summed weight that satisfies the C1P using a branch-and-bound algorithm.

`C1P/C1P_make_C1P_branch_and_bound.py`: Computes a subset of ACS that satisfies the C1P using a greedy heuristic.

`C1P/C1P_make_circC1P_branch_and_bound.py`: Computes a subset of ACS of maximum summed weight that satisfies the circ-C1P using a branch-and-bound algorithm.

`C1P/C1P_make_circC1P_branch_and_bound.py`: Computes a subset of ACS that satisfies the circ-C1P using a greedy heuristic.

`C1P/C1P_make_mC1P_branch_and_bound.py`: Computes a subset of ACS of maximum summed weight that satisfies the mult-C1P using a branch-and-bound algorithm. This option is computationally costly and is not available through the ANGES interface.

`C1P/C1P_make_mC1P_branch_and_bound_both.py`: Computes, from the non-telomeric ACS, a subset of ACS of maximum summed weight that satisfies the C1P using a branch-and-bound algorithm, then computes a maximum subset of telomeric ACS that augments it while satisfying the mult-C1P, again with a branch-and-bound.

`C1P/C1P_halve_PQRtree.py`: Given a PQR-tree or PQCR-tree with doubled markers, computes the corresponding tree with pairs of doubled markers reunited into a single, oriented, initial marker.

`SERIATION/SERIATION_compute Hamming_matrix.py`: Computes a Hamming distance matrix from a set of ACS with X-markers, as defined in [7].

`SERIATION/SERIATION_compute_PQtree_correlation_matrix.py`: Computes a PQ-tree from a set of X-markers free ACS using the method described in [1].

`SERIATION/SERIATION_compute_PQtree_dotproduct_correlation_matrix.py`: Computes a PQ-tree from a set of ACS with X-markers using the method described above: a correlation matrix corresponding to the dot product of the ternary matrix where X are replaced by a real number α is computed, then a PQ-tree is computed as in [1].

References

[1] J.E. Atkins, E.G. Boman, B. Hendrickson. A Spectral Algorithm for Seriation and the Consecutive Ones Problem. *SIAM J. Comput.* 28:297–310, 1998.

[2] A. Bergeron, C. Chauve, F. de Montgolfier, M. Raffinot. Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs. *SIAM J. Discrete Math.* 22:1022–1039, 2008.

[3] C. Chauve, J. Maňuch, M. Patterson, R. Wittler. Tractability results for the consecutive-ones property with multiplicity. In *CPM 2011*, *Lecture Notes in Comput. Sci.* 6661:90–103, 2011.

[4] C. Chauve, H. Gavranovic, A. Ouangraoua, E. Tannier. Yeast ancestral genome reconstructions: the possibilities of computational methods II. *Journal of Computational Biology* 17:1097–1112, 2010.

[5] C. Chauve, E. Tannier. A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes. *PLoS Computational Biology* 4(11):e1000234, 2008

[6] M. Dom, J. Guo, R. Niedermeier. Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. *J. Comput. Syst. Sci.* 76:204–221, 2010.

[7] H. Gavranovic, C. Chauve, J. Salse, E. Tannier. Mapping ancestral genomes with massive gene loss: A matrix sandwich problem *Bioinformatics* 27:i257–i265, 2011.

[8] J. Ma *et al.* Reconstructing contiguous regions of an ancestral genome. *Genome Res.*, 16:1557–1565, 2006.

[9] R. McConnell. A certifying algorithm for the consecutive-ones property. In *SODA 2004*, pp. 768–777, 2004.

[10] M. Muffato. Reconstruction de génomes ancestraux chez les vertébrés. PhD Thesis. University Évry Val d'Essonne. 2010

[11] M. Muffato, A. Louis, C.-E. Poisnel, H. Crollius. Genomicus: a database and a browser to study gene synteny in modern and ancestral genomes. *Bioinformatics* 26:1119–1121, 2011.

[12] A. Ouangraoua, E. Tannier, C. Chauve. Reconstructing the architecture of the ancestral amniote genome. *Bioinformatics* 27:2664–2671, 2011.

[13] T. Schmidt, J. Stoye. Quadratic Time Algorithms for Finding Common Intervals in Two and More Sequences. In *CPM 2004*, *Lecture Notes in Comput. Sci.* 3109:347–358, 2004.

[14] N. Vuokko. Consecutive Ones Property and Spectral Ordering. In *SIAM International Conference on Data Mining, SDM 2010*, pp. 350–360, 2010.